REMARKS

The applicant has carefully considered the Examiner's comments in his Office Action dated April 20, 2005, and respectfully traverses the rejections of the claims. The applicant submits that the invention as claimed is patentably distinguishable over Arvind et al.

First, the applicant submits that it is unnecessary to specify the meaning of "synchronous" in the claims in order to distinguish the invention over Arvind. The meaning of "synchronous parallel processing" is defined by the art and not by the present specification, and Arvind does not teach anything relating to synchronous parallel processing.

However, claim 1 is readily distinguishable over Arvind in other respects. The Examiner asserts in paragraph 6 of the Office Action that the token in Arvind is the instruction, and therefore claim 1, step a. is satisfied because a token destined for a dyadic operator that is received by the Wait-Match Unit (WM) before its other operand arrives is deposited in the WM to wait for the other operand (Arvind, p. 314, col 1). But the token in Arvind is not an instruction, it is an *operand*, which is data; in each case the instruction itself must be fetched from the Instruction-Fetch Unit in order for the operand (or operand pair) to be processed (Arvind, p. 314, col 1-2).

Nowhere does Arvind talk about a token being an instruction. Rather, they define "instructions" as "operators," not operands; and only data values between operators are carried on tokens (p.303, col. 1). Therefore, the claimed step a. "distributing at least one instruction for data processing to one data processing unit...before the data processing unit is available to process the instruction" is *not* met by Arvind. His system distributes a *data token* to a data processing unit before the data processing unit is available to process the data, but the instruction must be fetched from a common Instruction-Fetch Unit when the data is ready to be processed. The Wait-Match Unit never receives the instruction until all data is there ready to be processed.

Arvind teaches that Tagged-Token Dataflow Architecture (TTDA) is "a machine with purely data-driven instruction scheduling, unlike the sequential program counter-based scheduling of von Neumann machines" (Arvind, page 300, right column, paragraph 3).

von Neumann machines

Conventional von Neumann machines conceptually work as follows: Instruction are scheduled

according to an instruction (or program) counter. The processor fetches data packets according to addresses recorded in the instructions, waits for the data packets to arrive, executes the data packets, and sends resulting data packets out according to destination addresses recorded in the instructions. Performance stalls in a processor of such architecture occur because the processor is idle while waiting for the data packets arrivals.

Arvind et al.

On the other hand, purely data-driven machines, such as the TTDA processor described by Arvind, conceptually work as follows: Data tokens with corresponding data packets arrive at the processor. The processor processes the data tokens and fetches instructions according to the address recorded in the data tokens, waits for the instruction's arrival, executes the data packets according to the instructions to create new data tokens, and sends the resulting data tokens with corresponding data packets out according to the destination addresses recorded in instructions. Performance stalls in a processor of such architecture occur as in von Neumann machines, except that while the von Neumann processor is idle while waiting for the data packets to arrive, a purely data-driven processor is idle while waiting for the instructions to arrive.

Arvind describes in his paper how his Processing Element works (see Fig. 18 on page 313 and description on page 314), as follows: Data tokens with corresponding data packets arrive at Wait-Match Unit (WM). If the data packet is for a monadic operator it goes directly to the next stage, to the Instruction-Fetch Unit. If the data packet is for a dyadic operator it waits inside the WM for arrival of its pair before going to the next stage. The Instruction-Fetch Unit reads the address of the matching instructions from the data token (a couple of data tokens for a dyadic operator must refer to the same instructions), requests instructions from the memory, and waits for the instruction's arrival. Then data packets with instructions are then passed to the next stage, to the ALU and Compute-Tag Unit which performs operations on the data packets and produces resulting data packets. The Compute-Tag Unit also computes destination addresses for the resulting data tokens, based on destination address information recorded in the instructions. Then resulting data packets and calculated destination addresses are passed to next stage, to a Form-Tokens Unit that combines the data packets and destination addresses to form resulting data tokens, which are sent out according to the calculated destination addresses.

Thus, in Arvind the *data tokens* are distributed to the data processing units before the *instructions* arrive, and the instructions are fetched when the data token (for a monadic operator) or pair of data tokens (for a dyadic operator) are received by the Wait-Match Unit.

The Invention

In contrast, the present invention uses a mixed architecture, where instructions are processed (or scheduled) using counters as in conventional von Neumann machines; and data packets are processed (or executed) using data tokens, similar to "purely" data-driven machines. In order to achieve this the invention utilizes two separate paths – an Instruction Path and a Data Path.

The processor of the invention conceptually works as follows: Instruction are scheduled according to an instruction counter. The processor fetches data packets according to addresses recorded in the instructions, continues processing the instructions while keeping records of outstanding instructions in internal memory (as opposed to waiting for data packets to arrive), receives data packets with matching data tokens, executes the data packets according to directions recorded in the outstanding instructions held in internal memory, creates new data tokens for the resulting data packets, sends the resulting data packets with matching data tokens out according to destination addresses recorded in instructions, and erases the corresponding records from the records of outstanding instructions.

Because the invention uses two separate paths for instructions and data packets, it has a substantially non-stalling processor architecture. The processor of the invention doesn't wait for data packets to arrive after instructions scheduling, as in conventional von Neumann machines – the scheduled instructions are kept internally inside processor while processor continues instructions scheduling. Further, the processor of the invention doesn't wait for instructions to arrive after data token processing, as in purely data-driven machines – the scheduled instructions are kept as records of outstanding instructions and are already inside the processor when data packets and data tokens arrive to be processed.

Claim 1

In short, Arvind teaches a TTDA architecture, which is very different from the present invention. Conceptually the TTDA computational scheme consists of the following steps:

- distributing a data token containing the data packet to the processing unit;

- processing the data token;

- sending out a request for instructions according to addresses recorded in the data token;

- waiting for the instructions to arrive;

- executing data packets according to the instructions and creating new data tokens for resulting data packets; and

- sending out data tokens containing the resulting data packets according to destination addresses recorded in the instructions.

Comparing these steps with claim 1:

- Claimed step (a) is not met, since in the invention instructions arrive at the processing unit before the data packets, whereas in the TTDA architecture they arrive after the data packets.
- Claimed step (b) is not met, since the TTDA architecture doesn't have an execution instructions storage in the processing unit, and instructions are necessarily executed immediately after they are fetched (data packets are already inside the processing unit and waiting).
- Claimed steps (c) and (d) are not met, since the TTDA processor sends out requests for instructions and not for data packets.
- The order of steps as claimed is not met by the TTDA architecture.

The applicant accordingly submits that claims 1 to 11 are allowable over Arvind.

Claim 18

The applicant similarly traverses the Examiner's rejection of Claim 18 to 28 as being anticipated by Arvind. The Examiner is not giving due consideration to the fact that if two elements are recited in a claim they are considered to be separate elements, and the provision of separate data and instruction paths in the invention is a patentable distinction over Arvind. The applicant has in

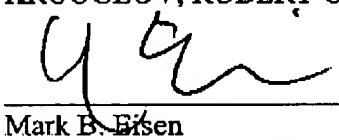
any event amended claim 18 to recite "a data path, separate from the instruction path, contained inside the processor," which as noted above is a patentable distinction over Arvind.

A Petition for an Extension of Time requesting an extension of one month for filing the subject response is enclosed. The Commissioner is authorized to charge any deficiency or credit any overpayment in the fees for same to our Deposit Account No. 500663. A signed copy of this page is enclosed if required for this purpose.

Favourable reconsideration and allowance of this application are respectfully requested.

Executed at Toronto, Ontario, Canada, on August 18, 2005.

DANIEL GUDMUNSON, ALEXEI KROUGLOV, ROBERT COLEMAN



Mark B. Eisen
Registration No. 33,088
(416) 971-7202, Ext. 242
Customer Number: 38735

MBE:lf
Encl. Duplicate of signature page
Petition for Extension of Time (in duplicate)